ACEEE Int. J. on Information Technology, Vol. 3, No. 1, March 2013

An Efficient Search Engine for Searching Desired File

Umesh Chandra Jaiswal[1], Rohit Kamal Saxena[2], and Kamlendra Pratap Singh[3]
[1] Madan Mohan Malaviya Engineering College, Gorakhpur, India
[2]Cisco Systems, Bangalore, India
[3]Bharat Electronics Limited, Gaziabad, India
Email:[1] ucj_jaiswal@yahoo.com, [2] rohit_kamal2003@yahoo.com,[3] 22kamlendra@gmail.com

***Abstract*—With ever increasing data in form of e-files, there always has been a need of a good application to search for information in those files efficiently. This paper extends the implementation of our previous algorithm in the form of a windows application. The algorithm has the search time-complexity of Θ(n) with no pre-processing time and thus is very efficient in searching sentences in a pool of files.**

***Index Terms*—Text search, sentence searching, searching in files application**

I. Introduction

In this 21st century, everything is getting documented day by day. We are piling up lots and lots of files that give rise to a need of good text searching applications. We have very few efficient applications that can search within files. The description of the application presented uses 'A Fast Sentence Searching Algorithm' for searching text/sentences in the files [8]. The main focus of the application is to search any sentence in the given pool of files in various folders or drives so that desired file can be searched on the basis of given information in the form of a sentence or a small paragraph. There exists various text searching algorithms like KMP, Boyre-Moore which can be efficient in case of patterns but while searching for sentences the performance of our algorithm is better than the rest of the algorithms, so it has been chosen for the application [1], [2], [3] and [4].

II. Related Work

Amongst the several text-searching algorithms designed until now, the simplest one is the Naive or Brute-Force Algorithm. Rabin-Karp is another searching technique that makes use of elementary number-theoretic notations such as equivalence of two numbers modulo a third number. Other algorithm is the Knuth-Morris—Pratt algorithm that is a linear time string-matching algorithm [5]. This algorithm uses a prefix function π that encapsulates knowledge about how the pattern matches against shifts or itself. Now the most commonly used text-searching algorithm is the Boyre-Moore Algorithm that takes a *sub-linear* searching time [6]. It uses two functions i.e. a bad character and a good prefix functions require certain preprocessing. Let *m* be the length of the sentence and let *n* be the length of the search space (file). The Table-I gives the comparison of the asymptotic time analysis of various text searching algorithms. A very little work has been performed in this area. There is no general purpose system available till today that provides the facility of searching desired file on the basis of information available [7], and [8].

Table I. Comparative AsymptoticTime Analysis

S. No.	Algorithm	Preprocessing time	Matching time
1	Naïve String Search Algorithm	0(no preprocessing)	Θ((n-m+1) m)
2	Rabin-Karp string search algorithm	Θ(m)	average Θ(n+m), worst Θ((n-m+1) m)
3	Finite state automaton based	Θ(m |Σ|)	Θ(n)
4	KMP algorithm	Θ(m)	Θ(n)
5	Boyer–Moore search Algorithm	Θ(m + |Σ|)	Ω(n/m), O(n)
6	*Our Algorithm*	0(no preprocessing)	Θ(n)

III. Algorithm

The algorithm used in the application for searching some sentence/search:

1. While(!EndOfFile)
2. Do read a single character from file, x
3. pos ← pos+1
4. If sentence[i]=x then
5. i ← i+1
6. Else
7. i ← 0
8. If sentence[i]=x then
9. i ← i+1
10. If i = LengthOfSentence then
11. c ← c+1
12. i ← 0
13. Return c

The above algorithm returns 'c', i.e. the number of times the sentence to be searched occurs in a single file. It can scan more than one file; one by one and thus help in distinguishing between the set of files that contain a sentence or paragraph and the ones that do not contain it.

The above algorithm works by scanning the file character by character and comparing each character of the file with the ones in the sentence to we wish to search. We may see the algorithm in two phases as described in the following lines.

(a) Initially, we compare the first character of the file is with the first character of the sentence to be searched. If there is a match, we increment i else we set the pointer again to 0 and check for the first character of the sentence.

(b) We now check the value of 'i' if it is equal to length of the sentence or not. Value of 'i' will be equal to the length of

DOI: 01.IJIT.3.1.1028

sentence only in a condition if the sentence is found thus we increment the value of 'c'.

IV. Theoretical Analysis

Considering the illustrated algorithm, we can see that the complete complexity of searching a sentence in a file is equal to Θ(n) without having any pre-processing time where n is the number of characters in the file.

Lines 1-12 show that this particular loop continues until the end of file i.e. iterates 'n' (no. of characters in file) times. Line 2 reads a single character at a time thus having O(1) complexity. Similarly, line 3 also executes once in a loop. Lines 4-9 check whether the character read from file is present in the sentence or not and accordingly, the respective lines execute. In case we find a mismatch, we check it for the first character of the sentence we are searching. Lines 10-12 check if the sentence is found in the file and accordingly increment the counter of the number of sentences by 1. Finally, line 13 returns the number of times the sentence is present in the file. This clearly shows that there is a single loop iterating 'n' times. Thus, the complexity of the algorithm is Θ(n), under all circumstances as the loop continues till the last character of the file whether or not the sentence is present in the file [5], [6], [7], and [8].

V. Implementation

The algorithm has been implemented in C#.NET using Visual Studio as IDE. There is an option of a single file or a complete folder that allows you to quickly search **inside** the files on the drive or network. It can easily retrieve the documents that contain the multiple sentences and phrases that one is interested in. Figure 1 shows the snapshot of the application's working. The list of files containing the sentence will be displayed that can redirect to the file on clicking it. The search can be performed on PDF, DOC, TXT, HTML and PPT files. Some of the extra features that have been included in our application are

Normal Searching:

Normal Searching allows the use of the question mark (?) and asterisk (*) to match one and one or more characters respectively. All white space is treated the same and multiple white space characters are treated as one.

Search a drive, path or multiple drives and paths:

Such as *C:\ | \\Corp-backup\C\Accounting*

Exclude specific folders or paths:

C:\ | -Windows | -Program Files

This option would exclude the folders *Windows* and *Program Files* and all their subfolders

Restrict to specific file types and patterns:

**.doc | *.rtf*

This would check only files with names ending in *doc* or *rtf*.

Exclude specific extensions:

-.bak | -*.tmp | -~**

This option would search for all files except those that have the extension *bak,tmp* or that start with the tilde character.

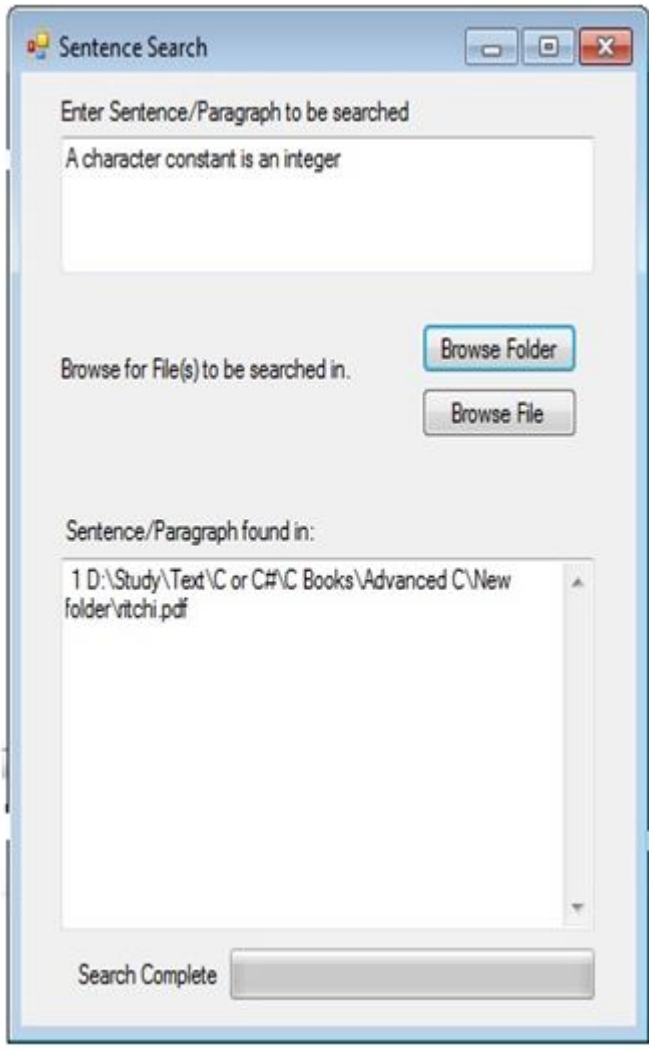

Fig. 1. Application Screenshot

Conclusion And Future Scope

With increased use of computer for documenting almost everything, we need such applications that can help searching in those documents. This application can be very useful as there are very few applications that solve this purpose. Secondly, since it uses efficient algorithm for searching, the results are computed at a very faster rate and thus saving a lot of useful time.

There will be a revolutionary change in the working of various offices of different organization. This will provide a user friendly way to search desired file or files on little information available from various media. Sometimes people are not able to find the desired files as the number of files becomes very large and spread in various folders. A natural language interface to the system may be developed so that it will be more users friendly in the offices of various organizations. We have a plan to integrate the developed search engine with English language. Later on this search engine may be extended for Hindi language files along with Hindi language interface.

DOI: 01.IJIT.3.1.1028

REFERENCES

[1] Cormen, T.H., Leiserson, C.E., Rivest, R. L., Stein, C. In: Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Chapter 32: String Matching, pp.906–932.

[2] Cole. R. "Tight Bounds on the complexity of the Boyer-Moore Algorithm". In: Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms.

[3] Rohit Kamal Saxena, U C Jaiswal, and Kamlendra PratapSingh, "A Fast sentence Searching Algorithm". In proceedings of the CNC-2011 (Second international Conference on Advances in Communication, Network, and Computing. The proceeding is on digital media. The conference is held during March 10-11, 2011 in Bangalore,India

[4] Karp, Richard M.; Rabin, Michael O. (March 1987). Efficient randomized pattern-matching algorithms.

[5] Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching strings. TR CS-74-440, Stanford University, Stanford California (1974).

[6] Boyer, R. S., Moore J. S.: A Fast String Searching Algorithm. In: Carom. ACM 20, (10), 262–272(1977)

[7] Daniel M. Sunday. 1990. A very fast substring search algorithm. Commun. ACM 33, 8 (August 1990), 132-142. DOI=10.1145/79173.79184

[8] Algorithms by Robert Sedgewick Addison-Wesley Publication Company. ISBN O-201 -06672-6. Chapter 19: String Searching, pp 241-256.

DOI: 01.IJIT.3.1.1028